Rapid Trajectory Optimization Using C-FROST with Illustration on a Cassie-Series Dynamic Walking Biped

Ayonga Hereid[1], Omar Harib[2], Ross Hartley[2], Yukai Gong[2], and Jessy W. Grizzle[2]

Abstract— One of the big attractions of low-dimensional models for gait design has been the ability to compute solutions rapidly, whereas one of their drawbacks has been the difficulty in mapping the solutions back to the target robot. This paper presents a set of tools for rapidly determining solutions for "humanoids" without removing or lumping degrees of freedom. The main tools are: (1) C-FROST, an open-source C++ interface for FROST, a direct collocation optimization tool; and (2) multi-threading. The results will be illustrated on a 20-DoF floating-base model for a Cassie-series bipedal robot through numerical calculations and physical experiments.

I. INTRODUCTION

Low-dimensional inverted pendulum models are by far the most common approach in the literature to getting around the analytic and computational challenges imposed by today's humanoid robots [18], [24]. In this approach, one maps a locomotion task onto the motion of the center of mass of a linear inverted pendulum (LIP), a spring-loaded inverted pendulum (SLIP), or other simplified models [22] and equips the model with a "foot-placement strategy" for stability [19], [25], [26]. The overall simplicity of this approach is one of its uncontested advantages. On the other hand, when designing agile motions, guaranteeing stability in the full-order model and exploiting the full capability of the machine, especially in light of physical constraints of the hardware or environment, are among its main drawbacks.

Because of these drawbacks, numerous groups are pursuing systematic approaches to utilize a high-dimensional mathematical model of a legged robot when designing dynamic behaviors and also when designing controllers that are sufficiently robust to realize these behaviors on the physical hardware. Speaking first to dynamic gait design, there is an increasing trend in the development of optimization-based planning algorithms which directly utilize more complicated dynamic models [4], [9], [17], [21]. These applications typically use general-purpose nonlinear programming solvers, e.g., IPOPT [31] or SNOPT [10], to formulate the motion planning problems. Others are advocating state-of-the-art optimal control toolboxes, such as GPOPS [23], DIRCOL [30] or PSOPT [3], that come with advanced trajectory optimization algorithms. These latter packages, however, either lack an effective framework for users to construct the robot model

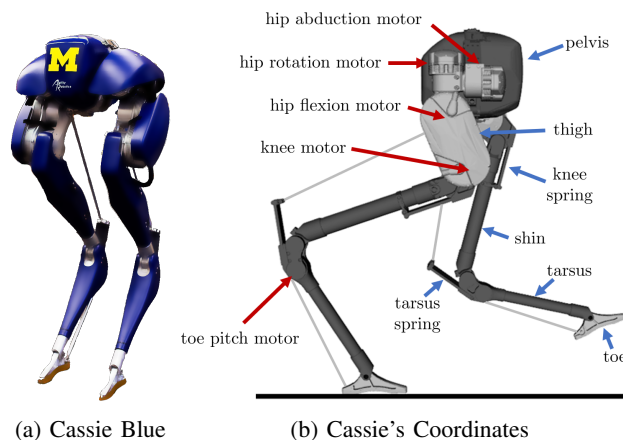(a) Cassie Blue   (b) Cassie's Coordinates

Fig. 1: (a) Shows Cassie Blue, one of the Michigan copies of the Cassie series of robots built by Agility Robotics. (b) Shows the joints and the kinematic model of the robot.

or are not capable of solving large-scale problems that are very common for today's agile walking robots.

More recently, several open-source robotic toolboxes, including DRAKE [29] and FROST [15], have been developed to provide an integrated modeling, planning, and simulation environment for complex robotic systems. Both DRAKE and FROST are capable of optimizing dynamic walking gaits for high degree of freedom (DOF) robots using the full-order natural dynamic models of the machines. While DRAKE is implemented in a C++ environment for efficiency, FROST is developed as a MATLAB package for rapid prototyping. Compared to DRAKE, which uses automatic differentiation for computing gradient information for the NLP solvers, FROST takes advantages of the symbolic computation to generate the sparsity structure of the trajectory optimization problem. To speed up the computational speed for high-dimensional robotic systems, FROST is equipped with a custom symbolic math toolbox based on the Wolfram Mathematica kernel as the back-end to generate optimized C++ code for computing system dynamics and kinematics functions. Moreover, FROST is capable of computing the analytic derivatives of optimization constraints and cost functions symbolically, and *even more importantly, it can precisely determine the sparsity structure of the nonlinear trajectory optimization problem*. With these features, FROST generates dynamic biped walking gaits using the full-order dynamics for a full-size humanoid robot (e.g., DURUS) in less than 10 minutes [16].

Since its inception, FROST has been used for fast dynamic gait generation of various bipedal robots [1], [14], [27]. Yet, its performance is significantly affected by the computational overhead introduced by MATLAB. This becomes evident particularly in the case when many different gait trajectories are required to be designed [7].

In this paper, we present C-FROST, an open-source package that converts the direct collocation trajectory optimization problem constructed in FROST to a more computationally efficient C++ code. By reducing the computational overhead of MATLAB functions, we are able to increase the speed over previous full-order dynamic optimization problems by a factor of six to eight for a single gait. The implementation of C-FROST also makes it easier to deploy parallel computation on cloud servers for the generation of a large set of different gaits, a feature that allows the gait library approach presented in [7] to be practical on high-dimensional 3D bipedal robots and advanced exoskeletons [14]. Beyond generating families of gaits for various walking speeds, directions, and terrain slopes, the ability to generate gaits in parallel for a family of models is also useful for iterative design of legged mechanisms [6], [27].

The remainder of the paper is organized as follows. Section II overviews the FROST package and the optimization problem being solved here for gait design. Section III introduces a stand-alone open-source C++ package that carries out the optimization problem formulated in FROST. Section IV presents a case study of the application of C-FROST for a Cassie robot, and its execution speeds are illustrated in Section V in terms of:

- optimization in FROST vs. C-FROST,
- serial execution vs. parallel, and
- desktop vs. the cloud.

Section V also demonstrates that the gait designs performed in the paper are viable on a Cassie robot. Conclusions are given in Section VI.

II. TRAJECTORY OPTIMIZATION IN FROST

FROST (Fast Robot Optimization and Simulation Toolkit) is an open source MATLAB toolbox that provides an integrated development framework for mathematical modeling, trajectory optimization, model-based control design, and simulation of complex robotic systems. While initially developed particularly for bipedal robots, FROST has been applied to a wide range of robotic systems, including quadrupeds [13], robot manipulators [20], and the adaptive lane keeping control design of an automated truck model [5]. The most important feature of FROST is the fast trajectory optimization algorithm for high-dimensional hybrid dynamical systems using direct collocation approaches [15].

A. Robot Behavior Modeling

In FROST, a particular behavior of a robot can be modeled as a hybrid system that exhibits both continuous and discrete dynamics. For instance, legged locomotion often consists of a collection of continuous phases, with discrete events triggering transitions between neighboring phases [12].

Definition 1. A *hybrid system* is a tuple,

$$\mathscr{HC} = (\Gamma, \mathcal{D}, \mathcal{U}, S, \Delta, FG), \tag{1}$$

where $\Gamma = \{V, E\}$ is a *directed graph*, $\mathcal{D}$ is the admissible configuration of the continuous domains, $\mathcal{U}$ represents the admissible controllers, S determines guard conditions that trigger discrete transitions, and Δ and FG represent the discrete and continuous dynamics respectively [15].

In a hybrid system model, each vertex represents an admissible continuous phase (a.k.a. domain) determined by the differential equation of motion and physical constraints of the robot. These constraints include the contacts or mechanical constraints such as four-bar links or contacts with the environment, which modeled as *holonomic constraints*, and associated limiting conditions such as the friction cone or swing foot clearance conditions, which can be formulated as a set of *unilateral constraints*. A *holonomic constraint* establish a set of algebraic equations to the continuous system dynamics. Moreover, for each edge in the graph, a *unilateral constraint* determines the guard condition that triggers the associated discrete event. For example, in legged locomotion, the robot switches to different continuous domains when it establishes new contacts or breaks existing contacts. FROST describes the discrete behaviors of a robot by defining these kinematic and dynamic elements of the robot in the framework of a hybrid system model given in (1).

Continuous Dynamics. The continuous domain describe the evolution of system states governed by differential algebraic equations (DAEs) on a smooth manifold. Let $x \in \mathcal{Q}$ be a set of coordinates of the system, the continuous dynamics is governed by the following ordinary differential equations (ODEs):

$$M(x)\ddot{x} = F(x, \dot{x}) + G(x, u) \tag{2}$$

subject to a set of algebaric equations $h(x) = 0$, where $M(x)$ is the positive definite mass matrix, $F(x, \dot{x})$ is a set of drift vectors, and $G(x, u)$ is a set of input vectors with u being the system inputs. FROST also allows to use a first-order ODEs to describe the system equations of motion [5], [15].

Discrete Dynamics. When the evolution of a system transitions from one continuous phase into another, its states often undergo an instantaneous discrete change. It can be given expressed in the following manner:

$$x^+ = \Delta_x(x^-), \tag{3}$$

where $x^- = \lim_{t \nearrow t_0} x(t)$ and $x^+ = \lim_{t \searrow t_0} x(t)$ with t_0 be the time instant at which the discrete dynamics occurs, and Δ_x represents a reset map at transition. For a second-order system, the first order derivative of x should also be considered, i.e.,

$$\dot{x}^+ = \Delta_{\dot{x}}(x^-)\dot{x}^-, \tag{4}$$

where $\dot{x}^- = \lim_{t \nearrow t_0} \dot{x}(t)$ and $\dot{x}^+ = \lim_{t \searrow t_0} \dot{x}(t)$.
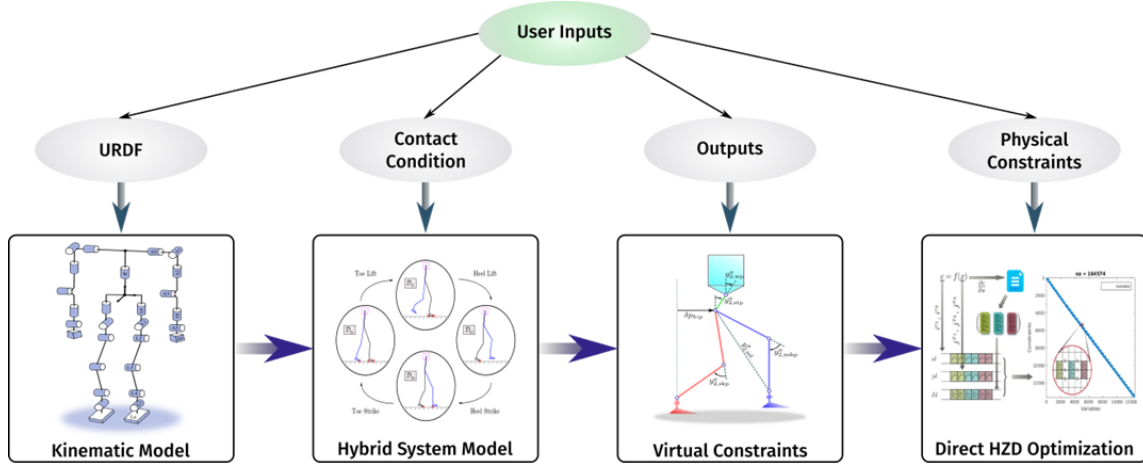
Fig. 2: The process of modeling and optimizing dynamic walking gaits in FROST. Given a robot model (e.g., via a URDF file), FROST makes it easy for users to define the desired behavior and provide the necessary physical constraints to formulate a realistic gait optimization problem.

Remark 1. Given a robot's kinematic structure and inertial parameters, FROST provides many useful functions to easily compute the forward kinematics and dynamics of the system [15]. A typical process in FROST for the modeling and gait design of a bipedal robot is shown in Figure 2.

B. Trajectory Optimization

The key functionality of FROST is to provide a fast and scalable trajectory planning algorithm for the hybrid dynamical systems. Given a hybrid dynamical system model in (1), FROST automatically constructs a multi-phase hybrid trajectory optimization problem [16]. In particular, it uses direct collocation methods to solve such a trajectory optimization problem. FROST supports various types of direct collocation methods, including local collocation methods that use Hermite-Simpson or Trapezoidal schemes and global collocation methods that collocate at Legendre-Gauss-Lobatto (LGL) points. For the mathematical formulation of different collocation methods, we refer [16] to the readers for more detail. The result of such a trajectory optimization problem is a large-scale sparse nonlinear programming (NLP) problem. In general, it can be stated as:

$$\text{argmin} \sum_{j=1}^{P} \left(\sum_{i=1}^{N_j} w_i \mathcal{L}_j(x_i, \dot{x}_i, u_i) + E_j(x_0, u_0, x_{N_j}, u_{N_j}) \right)$$
$$(5)$$

$$\begin{aligned}
\text{s.t} \quad & M_j(x_i)\ddot{x}_i = F_j(x_i, \dot{x}_i) + G_j(x_i, u_i), & \text{(dynamics)} \\
& \delta_j(x_0, \dots, x_i, \dot{x}_i, \ddot{x}_i, \dots, \ddot{x}_{N_j}) = 0, & \text{(collocation)} \\
& (x_0^{j+1}, \dot{x}_0^{j+1}) = \Delta_j(x_{N_j}^j, \dot{x}_{N_j}^j), & \text{(reset map)} \\
& C_j(x_i, \dot{x}_i, u_i) \geq 0 & \text{(path constraints)}
\end{aligned}$$

where $\mathcal{L}_j(\cdot)$ represents the running cost and $E_j(\cdot)$ represents the terminal cost defined in the domain $j \in V$ which has P vertices, δ is the collocation constraints, Δ_j is the reset map associated with domain $(j, j+1) \in E$, and $C(\cdot)$ is

the collection of physical constraints, such as foot clearance, joint angle/velocity limits and torque limits, etc.

III. C-FROST IMPLEMENTATION

This section presents C-FROST[1], an open source package that allows the user to convert and run the trajectory optimization problem in a native C++ environment instead of MATLAB. By reducing the computational overhead in the MATLAB environment, C-FROST noticeably speeds up the computational performances. It also comes with a few added benefits such as:

- using multi-threading techniques to speed up the evaluation of function Jacobians; and
- making it easier to execute multiple optimization problems in parallel on multi-core processors; and
- allowing easier deployment of parallel optimization to remote cloud services.

A. Formulation of NLP Problems

The use of direct collocation converts all constraints as algebraic constraints. While in more traditional trajectory optimization or optimal control problem, the state trajectories must be obtained by integrating the differential equations of motion. This enables us to construct the optimization problem as a general purpose nonlinear programming (NLP) problem, stated as:

$$X^* = \underset{X}{\text{argmin}} \quad \sum_{i=1}^{N_o} \mathcal{L}_i(X_i) \qquad (6)$$
$$\text{s.t} \quad \mathcal{C}_j^l \leq \mathcal{C}_j(X_j) \leq \mathcal{C}_j^u \quad \forall j \in [1, \dots, N_c],$$
$$X^l \leq X \leq X^u,$$

wher $X \in R^{N_x}$ with N_x being the total number of optimization variables, N_o is the total number of cost functions with

---

[1]C-FROST is publically available at https://github.com/UMich-BipedLab/C-Frost.

each cost function $\mathcal{L}_i$ has dependent variables X_i, where X_i is a collection of components of X, N_c is the total number of constraint functions, each constraint $\mathcal{C}_j$ has dependent variables X_j (again, X_j is also a collection of components of X), and $\mathcal{C}_j^l$ and $\mathcal{C}_j^u$, and X^l and X^u, respectively, are the lower and upper bounds of $\mathcal{C}_j$ and X. We also denote $\mathcal{C} = [\mathcal{C}_1; \ldots; \mathcal{C}_{N_c}]$ as the vector of all constraints, and $\mathcal{C}^l$ and $\mathcal{C}^u$ as the lower and upper bound of $\mathcal{C}$, respectively.

A particular advantage of using FROST is its capability to compute the closed-form symbolic expressions of all constraints and cost functions and export them to optimized C++ codes for fast computation. This is realized by the custom designed MATLAB symbolic math toolbox in FROST powered by the Wolfram Mathematica kernel as the backend. While formulating the problem in (6), we introduce additional *defect variables* to simplify the constraints expressions in FROST. That enables FROST to compute the analytic derivatives (a.k.a., gradient or Jacobian matrices) of constraint and cost functions with respect to the corresponding dependent variables, as well as to determine the non-zero entries in the derivatives. In C-FROST, we generate two functions to compute the sparse Jacobian matrices: one that returns the row and column indices of all non-zero entries, and another one computes the values of these non-zero entries. Using these two functions, one can formulate the sparse Jacobian matrix of a constraint or the gradient of the cost function.

B. Execution of NLP Problems

C-FROST contains two main components: a collection of MATLAB functions that convert and export the original optimization problem in MATLAB to a set of C++ functions and JSON configuration files; and a C++ program that uses these exported functions and configuration files to run the optimization on a native Linux environment by calling the IPOPT solver directly.

Exporting the NLP Problem. To export an NLP problem to the C-FROST format, we extract the structure of the problem in (6) and export this information to a collection of JSON files. For each constraint, the following information is required:

- the associated C++ functions, f_j to compute the constraint or cost function, G_j to compute the non-zeros in the Jacobian matrix, and Gs_j to return the indices of non-zero terms;
- the indices of dependent variables X_j in X;
- the indices of constraints $\mathcal{C}_j$ in $\mathcal{C}$;
- the auxiliary constant data `aux` required to compute f_j;
- the indices of non-zero Jacobian entries G_j in G_{nz}.

The above information is constructed as arrays of structured data, `Constraints`, and exported to a JSON file. The same information for the cost functions is also formulated and exported to the same JSON file. We also generate two additional configuration files with one storing the boundary values of constraints and optimization variables and another storing the initial guess for the NLP solver. The C++ functions f_j, G_j and Gs_j are generated automatically using the symbolic math toolbox in FROST.

Procedure 1 Evaluation of the NLP Constraints

1: **procedure** EVALFUNC(X, `Constraints`)
2:    Initialize $\mathcal{C} \leftarrow 0$
3:    **for all** $f_j \in$ `Constraints` **do**
4:       $x_j \leftarrow$ the dependent variables of f_j from X
5:       `aux` $\leftarrow$ the auxiliary constants of f_j
6:       $\mathcal{C}_j \leftarrow$ the evaluation of $f_j(x_j; $ `aux`$)$
7:       $r_j \leftarrow$ the indices of $\mathcal{C}_j$ values in $\mathcal{C}$
8:       $\mathcal{C}(r_j) \leftarrow \mathcal{C}_j$
9:    **end for**
10:    **return** $\mathcal{C}$
11: **end procedure**

Procedure 2 Evaluation of the sparse Jacobian matrix of NLP Constraints

1: **procedure** EVALJAC(X, `Constraints`)
2:    Initialize $G_{nz} \leftarrow 0$, $i_R \leftarrow 0$, $j_C \leftarrow 0$
3:    **for all** $f_j \in$ `Constraints` **do**
4:       $x_j \leftarrow$ the dependent variables of f_j from X
5:       `aux` $\leftarrow$ the auxiliary constants of f_j
6:       $J \leftarrow$ the evaluation of $G_j(x_j; $ `aux`$)$
7:       $(i_r, j_c) \leftarrow$ the evaluation of $Gs_j(x_j; $ `aux`$)$
8:       $r_j \leftarrow$ the indices of J values in G_{nz}
9:       $G_{nz}(r_j) \leftarrow J_i$, $i_R(r_j) \leftarrow i_r$, $j_R(r_j) \leftarrow j_c$
10:    **end for**
11:    **return** sparse(G_{nz}, i_R, j_C) ▷ Construct the sparse matrix
12: **end procedure**

Evaluating the NLP Problem. Using the generated C++ functions, C-FROST will create a C++ program that calls the interior-point-based NLP solver IPOPT as a shared object to solve the direct collocation optimization problem. The program will first read the exported configuration files to construct a problem for IPOPT, and then call the IPOPT optimization routine, which requires sub-routine functions that compute the cost function and its gradient, the constraints, and the sparse Jacobian of the constraints. In Procedure 1, we describe how the constraints are computed. The same procedure applies to the cost function. Procedure 2 illustrates the construction of the sparse Jacobian matrix for constraints, using the generated C++ functions and other indexing information. Again, the same procedure is used for calculating the gradient of the cost function. It is important to note that we will only need to compute the non-zero entries in the Jacobian matrix. Considering that the percentage of non-zero entries in the Jacobian of a FROST-generated NLP problem is often far less than 1%, this significantly reduces the overall computational load.

C. Parallel Evaluation of NLP Problems

To speed up the optimization further, we consider two types of parallelization in FROST. For a single gait optimization, C-FROST provides multi-thread parallelization for the evaluation of Jacobian, whereas to run multiple optimization

problems together, we prefer to run them in parallel on a multiple CPU cores platform.

Subroutine Parallelization. The evaluations of the Jacobian of some constraints might be computationally expensive, like in the case of the dynamics equation. Therefore there are benefits to parallelization when evaluating such matrices. The subroutines of each constraint within the for-loop in Procedure 2 are independent of other constraints; hence they can be run in parallel. C-Frost allows the different constraint Jacobians to be evaluated simultaneously by multi-threading Procedure 2. For each constraint f_j, lines 4 till 9 in Procedure 2 are evaluated in available workers. To avoid concurrent manipulation of the shared data G_{nx}, i_R and j_C, the evaluations in line 9 is locked using a monitor class to synchronize access to the shared data. The evaluation of the constraint Jacobian is significantly more computationally expensive than the evaluations of the constraint functions, the cost functions and the gradient of the cost function. And thus, these processes have not been parallelized.

Parallel Optimization. When having multiple optimization problems to be solved, as in the application of generating a library of gaits, running the multiple optimization problems simultaneously is preferred to using multiple threads per problem as that avoids the overhead from thread synchronization in a single program. This can be done by writing a simple bash script that starts new optimizations when a CPU is underutilized or existing third-party tools, such as the GNU Parallel package [28].

The ability to run many different gait optimization in parallel is particularly useful for learning-based control approaches as described in [7], [8], and can also be used in the mechanical design process of a robot, where instead of gait characteristics varying, link lengths, mass distributions, and gear ratios are varied [27], [32].

IV. A Case Study: Cassie Blue

The efficiency of C-FROST will be illustrated by designing a library of different periodic walking gaits for the Cassie-series bipedal robot shown in Figure 1.

A. Testbed Robot Model

This underactuated robot's floating base model has 20 degrees of freedom (DOF). There are seven joints in each leg with five of them actuated by electric motors through gearing and the other two joints being passive where they are realized via a specially designed four-bar linkage with one bar being a leaf spring. When supported on one foot during walking, the robot is underactuated due to the narrow feet. The springs in the four-bar linkages also introduce additional underactuation. For simplicity, we assume these springs have infinite stiffness, i.e, rigid links, in this paper (see Figure 1b). The description of the mechanical model of Cassie used in this paper is publicly released on GitHub[1].

In this paper, we will use FROST to model and formulate a series of periodic walking gait optimization problems for the Cassie, and then use C-FROST to generate these gaits in parallel rapidly. The optimization performance will be evaluated on multiple platforms—including remote cloud services—with different numbers of CPU cores. While the present paper is concerned with optimization for the open-loop model of Cassie, some of the designed gaits are implemented to demonstrate that they are physically realistic, and for that, a control policy is necessary. The control laws used in the simulations and experiments reported here are based on the work in [11]. To enhance our confidence that the presented optimization results are representative of what other users will find on their models, we have also run several of them on an 18-DOF exoskeleton model presented in [14]. These results are not reported here.

B. Optimization Problem

The primary problem being illustrated in this paper is to design two-step periodic walking gaits for Cassie Blue. This walking pattern is modeled as a two-domain (a right-stance domain and a left-stance domain) cyclic hybrid system model in FROST. A virtual constraints based feedback controller is also enforced in the gait optimization problem so that the results are not just the open-loop join trajectories but also a class of feedback controllers that can be used to stabilize the periodic motion in the full-order robot dynamics [16].

The optimization problems are formulated in FROST using the structure described in (6). When formulating the gait optimization problems, we impose periodicity over two steps for a desired average walking speed and ground inclination. The cost function to be minimized in the optimization is the "pseudo energy", given as

$$\mathcal{L}_j = \|u_i(t)\|^2, \tag{7}$$

where $\mathcal{L}_j$ is the integral function for the running cost defined at the domain j, with $j \in \{1, 2\}$ being the index of the right and left stance domain and u_i being the control inputs at each collocation node. The "pseudo energy" in (7) is not normalized by step length because the speed and step duration are constrained. In addition, the optimization also captures torque and joint limits of the Cassie robot as well as the following physical constraints that are heuristically chosen to achieve stable walking. These constraints are:

- Fixed step duration T of 0.4 s;
- Swing foot clearance of at least 15 cm at midstep;
- Ground reaction forces respect the friction cone and ZMP condition [12];
- Zero swing foot horizontal speeds at impact;
- For purely sagittal plane walking, require the two steps in a gait to be symmetric;
- Sagittal plane walking gaits have tighter bounds on hip abduction and rotation motors than gaits that include lateral motion; and
- Torso must remain upright within ± 3 degrees pitch and roll limits.

(a) 15 cm step up while walking forward at 1 m/s.

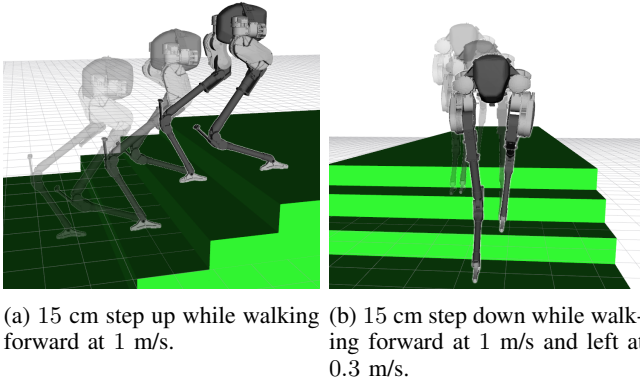(b) 15 cm step down while walking forward at 1 m/s and left at 0.3 m/s.

Fig. 3: Stereoscopic tiles for stepping up/down at different walking speeds.

In particular, we use C-FROST to design a library of $11^3 = 1,331$ periodic gaits defined on a cubic uniform grid with different walking speeds and ground inclinations:

- 11 different backward/forward walking speeds ranging from -1 m/s to 1 m/s;
- 11 different left/right walking speeds ranging from -0.3 m/s to 0.3 m/s; and
- 11 different inclinations of the ground surface from -15 cm to 15 cm per step.

The desired average walking speed is enforced via an equality constraint

$$\bar{v}_x = \frac{p_N^x - p_0^x}{T}, \quad \bar{v}_y = \frac{p_N^y - p_0^y}{T} \tag{8}$$

where p_N^x, p_0^x, p_N^y, and p_0^y are the (X,Y) position of the hip at the end and beginning of a step, respectively, and $\bar{v}_x$ and $\bar{v}_y$ are the desired sagittal and lateral walking speeds. The desired ground inclination will be enforced by changing the height where the swing foot should impact with the ground. Two of the gaits from this set are illustrated in Figure 3.

V. RESULTS

This section first presents the computational performance for creating a collection of periodic walking gaits for Cassie Blue under different scenarios using C-FROST[1]. Then we validate the feasibility of the optimized walking gaits on Cassie through experiments.

A. Computational Performance Analysis

To show how C-FROST can be used to improve the gait optimization process, we first compare its performance with the original MATLAB implementation in FROST, and then secondly, we show the improvement in performance when using parallelization on different platforms, including commercial cloud servers with large numbers of CPU cores.

Comparison versus FROST. In this comparison, we tested running a single gait optimization using FROST on MATLAB and using C-FROST running as a native program on

[1]The source code of this work is now publically available on https://github.com/UMich-BipedLab/Cassie_CFROST.
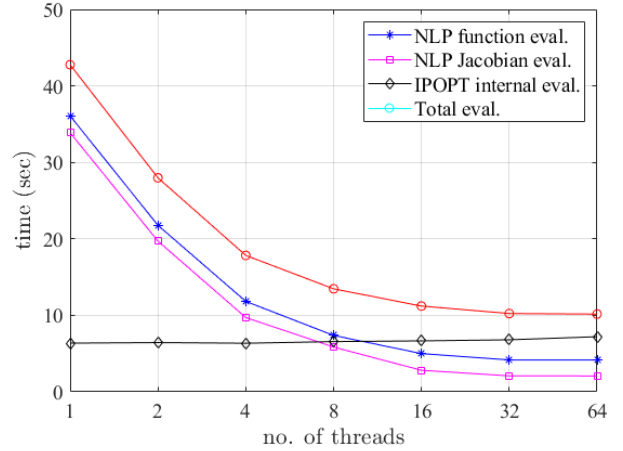

Fig. 4: The computation time of each components while using different number of threads.

Ubuntu. The optimization is set to design a walk-in-place gait (zero sagittal and lateral speeds) on flat ground. Both tests were run on a Thinkpad laptop with a 2.6GHz Intel i7 processor, under identical settings, and with the same initial guess. No parallelization was enabled in this test.

In our test, it took 90.12 seconds to generate the walk-in-place gait when running on MATLAB, whereas, using C-FROST resulted in the gaits being generated in 24.51 seconds. The result shows that using C-FROST speeds up the gait optimization process by approximately 3.5 times for a single gait. This demonstrates that C-FROST can be used to rapidly generate dynamic motions for legged robots without dealing with the computational overhead associated with MATLAB while still allowing one to use the easy-to-prototype environment provided by FROST. It should be noted that the comparison is the time it takes to run the optimizations and excludes the setup time to define the optimization problems since the latter is essentially the same for both methods.

Single gait optimization with subroutine parallelization. Further analyzing the result of C-FROST optimization, approximately 80% of the total computational time was spent to evaluate the NLP functions and the rest was used for IPOPT internal computation. Moreover, approximately 85% of the NLP function evaluation time was used to compute the sparse Jacobian matrix of the constraints. It indicates that computing Jacobian matrix becomes the main performance bottle neck.

Here, we report the performance improvements while enabling multi-threading for the computation of Jacobian matrix as described in Section III-C. In this test, we ran the walk-in-place gait optimization on a Linux virtual machine on the Google Cloud Platform. The results are shown in Figure 4. The plot shows that while the time required for IPOPT internal computation and other user-defined NLP functions other than the constraint Jacobian (the different between blue and purple line) remaining the same, the time to compute the constraint Jacobian (the purple line) reduces

TABLE I: Time to perform $11^3 = 1,331$ gait optimizations for the Cassie. The same pre-compiled code is run on all platforms. The average compute time for a gait is similar in all cases; the total times essentially inversely scale with the number of threads with slight differences with regards to the CPU frequency.

Computing Resource	CPU Cores	OS	Time (min)
Desktop 9-7900X @ 3.30GHz	10 cores (20 threads)	Ubuntu 18.04	401.18
Google GCP Xeon Skylake @ 2.0 GHz	64 vCPUs (64 threads)	Ubuntu 18.04	113.93
Amazon EC2 Xeon Platinum @ 3.0 GHz	72 vCPUs (72 threads)	AMI Linux 4.14	91.11

significantly when the available number of threads increases. It reduces from 34 (single core) to 2 seconds (64 cores), which yields a 17 times speed-up. When having more than 16 cores, however, the overhead time for synchronization between threads will dominate the overall run time, and the change is barely noticeable. Moreover, the affect on the total evaluation time becomes less when it requires more time for IPOPT internal computation. Overall, the total optimization evaluation sped up from 42.8 seconds to 10.2 seconds when using up to 64 cores.

Generating multiple gaits in parallel. The speed-up factor of the subroutine parallelization is not significant enough if considering the number of CPU cores being used. In need of generating multiple gaits at the same time, running all optimizations in parallel on available CPU cores will be more efficient. In our final test, all $1,331$ gaits were generated in a single batch by running the optimization solvers in parallel using all available cores on various computing platforms. The JSON configuration files that specify the boundary condition for each optimization are generated a priori, and then multiple instances of the same compiled program were set to run on different CPU cores using different boundary conditions under identical settings. When attempting to run multiple optimizations in parallel, we use the GNU Parallel package. Table I shows the computation times, specific machines, number of cores, and OS. The results show that utilizing all available computing resources of the given platforms, C-FROST can dramatically reduce the total time to compute batches of gaits. This feature is particularly important for learning based walking controllers, which require that one generate a large set of gaits as training data. In particular, users can take advantage of commercial cloud servers, such as Amazon's AWS or Google's GCP to generate a large set of gaits in a much shorter period. For example, using 72 vCPUs on AWS yields an average-per-gait computation time of approximately 4.1 seconds. This result shows that C-FROST noticeably reduces the computation time of trajectory optimization for a high-dimensional hybrid dynamical system without compromising problem formulation with simplified models.

B. Experimental Validation

The purpose of the experiments is to assure the reader that the set of constraints imposed during trajectory optimization are physically realistic and were not selected to simplify the designs. The controller that realizes the trajectories on Cassie Blue is based on [8] as detailed in [11]. Note that, only a
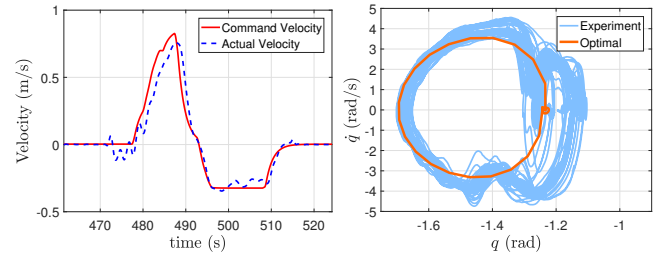
(a) The velocity tracking performance of Cassie walking.

(b) The phase portrait of the right knee joint during stepping in place.

Fig. 5: Experimental results for a gait-library-based controller on Cassie Blue.

subset of gaits from the previously generated 1,331 gaits are being utilized in the experiment. Regulating lateral velocity and ground inclination that uses the full set of gaits will be subject to future work.

During these tests, the 11 sagittal gaits with speeds ranging from -1.0 m/s to 1.0 m/s on flat ground were used to generate a gait library [8], [11] for Cassie Blue to regulate the forward and backward walking speeds. A heuristic foot placement controller was also applied to stabilize the lateral motion of the robot. In the experiment, we use a remote controller to command a reference speed profile, and the robot utilized the gait library to follow the commanded speeds. The robot can accelerate from 0 m/s to 0.8 m/s in around 10 seconds and then slow down to walking in place (e.g., 0 m/s) in 5 seconds. The robot is then commanded to walk backward with a nearly constant velocity of 0.3 m/s across the room to return to the starting point. Figure 5a shows the tracking of the commanded velocity in the experiment. Also, Figure 5b compares the similarity in the phase portraits of the right knee angle from optimization and experiment when the robot is stepping in place (i.e., zero forward velocity). A video that illustrates the different walking motions from the gait library built from optimization and the presented experimental results is available at [2].

VI. CONCLUSIONS

The primary objective of the paper was the presentation of a method to speed up trajectory design for high-dimensional systems by 3-4 times faster for single trajectories if not using parallelization and up to 9 times faster with parallelization. Moreover, the average computational time for a gait reduced significantly for families of trajectories. This was accom-

plished through the design of C-FROST, an open-source C++ interface for FROST—a MATLAB-based direct trajectory optimization tool for robotic systems. The significant speed up was illustrated by the design of a library of walking gaits for a Cassie bipedal robot. For a single gait, C-FROST completed an optimization in 24.51 seconds vs 90.12 seconds in FROST. The speedup is much more significant when doing multiple gaits, such as computing a (periodic) gait library [8] or building an invariant surface of open-loop transition trajectories [7]. The paper documents the parallel computation in C-FROST of $1,331$ gaits for a Cassie-series bipedal robot on a range of platforms. The average computation time is roughly 4.1 seconds per gait while running on an AWS virtual machine. In future work, we will consider using other parallel platforms, such as GPUs, to speed up the trajectory optimization problems further.

ACKNOWLEDGMENT

REFERENCES

[1] Cassie Blue. https://www.youtube.com/watch?v=oyKCBc9CZUA&list=PLFe0SMV3hBCBPg0dDG9FbqOy_x7H9kEDJ. Accessed: 2018-07-08.

[2] Video to Illustrate CFROST on the Bipedal Robot Cassie Blue. https://youtu.be/pweV7NLr6JY. Accessed: 2019-02-27.

[3] Victor M Becerra. Solving complex optimal control problems at no cost with PSOPT. In *Computer-Aided Control System Design (CACSD), 2010 IEEE International Symposium on*, pages 1391–1396. IEEE, 2010.

[4] Justin Carpentier, Steve Tonneau, Maximilien Naveau, Olivier Stasse, and Nicolas Mansard. A versatile and efficient pattern generator for generalized legged locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, May 2016.

[5] Yuxiao Chen, Ayonga Hereid, Huei Peng, and Jessy Grizzle. Enhancing the performance of a safe controller via supervised learning for truck lateral control. *arXiv preprint arXiv:1712.05506*, 2017.

[6] C. Chevallereau, G. Abba, Y. Aoustin, F. Plestan, E. R. Westervelt, C. Canudas-De-Wit, and J. W. Grizzle. Rabbit: a testbed for advanced control theory. *IEEE Control Systems*, 23(5):57–79, Oct 2003.

[7] Xingye Da and Jessy Grizzle. Combining Trajectory Optimization, Supervised Machine Learning, and Model Structure for Mitigating the Curse of Dimensionality in the Control of Bipedal Robots. *arXiv preprint arXiv:1711.02223*, 2017.

[8] Xingye Da, Omar Harib, Ross Hartley, Brent Griffin, and Jessy W Grizzle. From 2D design of underactuated bipedal gaits to 3D implementation: Walking with Speed Tracking. *IEEE Access*, 4:3469–3478, 2016.

[9] Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *2014 14th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 295–302. IEEE, 2014.

[10] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.

[11] Yukai Gong, Ross Hartley, Xingye Da, Ayonga Hereid, Omar Harib, Jiunn-Kai Huang, and Jessy Grizzle. Feedback control of a cassie bipedal robot: Walking, standing, and riding a segway. In *To be appear in American Control Conference (ACC)*, 2019.

[12] J. W. Grizzle, C. Chevallereau, R. W. Sinnet, and A. D. Ames. Models, feedback control, and open problems of 3D bipedal robotic walking. *Automatica*, 50(8):1955 – 1988, 2014.

[13] Kaveh Akbari Hamed, Wen-Long Ma, and Aaron D. Ames. Dynamically stable 3d quadrupedal walking with multi-domain hybrid system models and virtual constraint controllers. *arXiv preprint arXiv:1810.06697*, 2018.

[14] O. Harib, A. Hereid, A. Agrawal, T. Gurriet, S. Finet, G. Boeris, A. Duburcq, M. E. Mungai, M. Masselin, A. D. Ames, K. Sreenath, and J. W. Grizzle. Feedback control of an exoskeleton for paraplegics: Toward robustly stable, hands-free dynamic walking. *IEEE Control Systems Magazine*, 38(6):61–87, Dec 2018.

[15] A. Hereid and A. D. Ames. FROST*: Fast robot optimization and simulation toolkit. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 719–726, Sept 2017.

[16] Ayonga Hereid. *Dynamic Humanoid Locomotion: Hybrid Zero Dynamics based Gait Optimization via Direct Collocation Methods*. PhD thesis, Georgia Institute of Technology, 2016.

[17] Alexander Herzog, Nicholas Rotella, Stefan Schaal, and Ludovic Righetti. Trajectory generation for multi-contact momentum control. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 874–880. IEEE, 2015.

[18] S Kajita, F Kanehiro, K Kaneko, K Yokoi, and H Hirukawa. The 3D linear inverted pendulum model: a simple modeling for a biped walking pattern generation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 239–246, 2001.

[19] S. Kajita, T. Yamaura, and A. Kobayashi. Dynamic walking control of biped robot along a potential energy conserving orbit. 8(4):431–437, August 1992.

[20] S. Kolathaya, W. Guffey, R. W. Sinnet, and A. D. Ames. Direct collocation for dynamic behaviors with nonprehensile contacts: Application to flipping burgers. *IEEE Robotics and Automation Letters*, 3(4):3677–3684, Oct 2018.

[21] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the Atlas humanoid robot. *Auton. Robots*, 40(3):429–455, March 2016.

[22] Tad McGeer. Passive dynamic walking. *the international journal of robotics research*, 9(2):62–82, 1990.

[23] Michael Patterson and Anil Rao. GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Transactions on Mathematical Software (TOMS)*, 41(1):1, 2014.

[24] J. Pratt, T. Koolen, T. de Boer, J. Rebula, S. Cotton, J. Carff, M. Johnson, and P. Neuhaus. Capturability-based analysis and control of legged locomotion, part 2: Application to m2v2, a lower-body humanoid. *The International Journal of Robotics Research*, 31(10):1117–1133, August 2012.

[25] Jerry E Pratt and Russ Tedrake. Velocity-based stability margins for fast bipedal walking. pages 299–324. Springer, 2006.

[26] Marc H Raibert et al. *Legged robots that balance*, volume 3. MIT press Cambridge, MA, 1986.

[27] Jacob Reher, Eric A. Cousineau, Ayonga Hereid, Christian M. Hubicki, and Aaron D. Ames. Realizing dynamic and efficient bipedal locomotion on the humanoid robot DURUS. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[28] O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, Feb 2011.

[29] Russ Tedrake and the Drake Development Team. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2016.

[30] Oskar von Stryk. Users guide for DIRCOL version 2.1. *Simulation and Systems Optimization Group, Technische Universität Darmstadt, www. sim. informatik. tu-darmstadt. de/sw/dircol*, 1999.

[31] Andreas Wächter and T. Lorenz Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2005.

[32] E. R. Westervelt and J. W. Grizzle. Design of asymptotically stable walking for a 5-link planar biped walker via optimization. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 3, pages 3117–3122, 2002.